

## Assignment for Introduction

*Give the letter and number of the problem with your submission, and date your submission.  
For B-E, submit on paper and restate the problem.  
For all assignments, students may discuss solutions with each other.*

### **A. Self-introduction**

Post a brief introduction to yourself, at the “Introduction” forum of Blackboard Discussion Board. You are invited to mention your expectations and concerns in taking this course, your background with computers and IT, and your commitments, such as work, athletics, or family. You are invited to say what instructional methods help you learn best.

### **B. Basic skills**

Prepare to present in class your answer to the problem in the Introduction problem-solving quiz, corresponding to your classroom ID, or to another question listed in the study questions, Longer Answer.

### **C. Coding or pseudocode**

In the Study Questions, Longer Answer, under the heading “Algorithm design,” solve the problem that corresponds to your classroom ID. Prepare to present your answer in class.

### **D. Proof in propositional logic (optional)**

In the Study Questions, under the heading “Proofs in propositional logic,” solve the problem corresponding to your classroom ID.

### **E. Induction (optional)**

In the Study Questions, under the heading “Inductive proofs,” solve the problem corresponding to your classroom ID.

## Assignment for topic 1 (problem classes)

Each student will solve one of the problems in each set below corresponding to the student's classroom ID.

Give problem letter and number with solution, restate problem, and date your submission.

### A. Problem specification

Using the languages of predicate logic (with quantifiers, if appropriate) and algebra, give the formal *postcondition* for an algorithm (but not the algorithm itself) that accepts an array  $A$ , or arrays  $A$  and  $B$ , and returns value  $y$  as stated below.

For most problems, you will need to use the universal or existential quantifiers ( $\forall$ ,  $\exists$ ). Note that  $A$ ,  $B$  are arrays, not sets, so that for example, to claim that all elements of  $B$  are ones, we would write,  $(\forall i \leq |B|) B[i] = 1$ .

0.  $y = A$  in ascending order.
1.  $y =$  the bitwise AND of  $A$  and  $B$ , where  $A$  and  $B$  are bit vectors.
2.  $y =$  the bitwise OR of  $A$  and  $B$ , where  $A$  and  $B$  are bit vectors.
3.  $y = \text{true}$  iff any consecutive elements of  $A$  are the same.
4.  $y = \text{true}$  iff all elements of  $A$  are the same.
5.  $y = \text{true}$  iff any consecutive elements of  $A$  differ.
6.  $y = \text{true}$  iff any two elements of  $A$  are the same.
7.  $y =$  the maximum value in  $A$ .
8.  $y = \text{true}$  iff the number of 0's in bit vector  $A$  is greater than the number of 1's.
9.  $y =$  the sum of the elements of  $A$ .
10.  $y =$  the index of the first occurrence of key value  $x$  in  $A$ , or 0 if not found.

### B. Constraint vs. optimization

See Study Questions, topic 1, "Constraint vs. optimization." Solve the problem whose letter, converted to a number, corresponds to your classroom ID.

### C. Computation Tree Logic

Write and explain a CTL formula that applies to a reactive system and asserts, for formulas  $\phi$  and  $\psi$ , that on this system,

0.  $\phi$  will never hold.
1. there is a way for  $\phi$  to become true
2. after the next state transition,  $\phi$  will hold.
3. only after  $\phi$  holds can  $\psi$  cease to be true.
4. there is no way to get into the state  $\phi$ .
5. there is a way to get out of the state  $\phi$ .
6. there is no way to avoid getting into state  $\phi$ .
7. after some future state transition,  $\phi$  will hold.
8. after some future state transition,  $\phi$  will not hold.
9.  $\phi$  will hold as long as  $\psi$  is not true.
10.  $\phi$  or  $\psi$  will hold in all future states.

## Assignment for topic 2 (Verification)

*Each student will solve one of the problems in each set below. Groups may discuss solutions. Give problem letter and number, and restate problem, with your submission.*

*For ideas on how to solve these problems, refer to the handout, "Algorithm verification: sample problems."*

### A. Explaining loop invariants

See study questions, topic 2, "2b. Explain loop invariant." Solve the problem that corresponds to your classroom ID.

### B. Proofs using loop invariants

See problems 1-12 on Study Questions, Topic 2 "2b, 2c. proof of correctness." For the question that corresponds to your classroom ID,

- (a) Write appropriate postconditions and loop invariants for the pseudocode below, showing where the loop invariant should appear.
- (b) Use your assertions to argue that the pseudocode is correct.

### C. Correctness proof

See code or pseudocode you wrote for the assignment for Introduction, part C. Prove correctness of your solution using loop invariant.

0. Average
1. Subscript of the first element that is same as its successor
2. Number of elements starting with first, that are all the same
3. Smallest element
4. Length of longest ascending sequence starting with first element
5. Range (maximum minus minimum)
6. Number of zeroes
7. Subscript of largest element
8. Tell whether all values are the same
9. Sum of squares
10. Tell whether values are in ascending order

### D. (Extra credit) Formal proofs using proof rules

For the algorithm you wrote under problem set A above, write a correctness proof by writing after each line an assertion comment that follows from the previous assertion and the previous line of pseudocode. Each assertion should be accompanied with a reference to a proof rule (1. composition, 2. assignment, 3. *if*, or 4. *while*).

## Assignment for topic 3 (Recurrences; brute force)

Solve the problem corresponding to your classroom ID in each set below. Groups may discuss solutions. Give problem letter and number; restate problem.

### A. Big-O notation

See Study Questions, topic 3, Longer Answer. Solve a problem among those numbered 0-9 by explaining why the assertion about a big-O expression is true or false.

### B. Big-O and $\Theta$ notation

In plain English define these *sets of mathematical functions*, simplifying if necessary:

0. a.  $O(1)$       b.  $\Theta(n^2)$
1. a.  $O(\lg n)$     b.  $\Theta(n^3)$
2. a.  $O(n)$         b.  $\Theta(\lg n)$
3. a.  $O(n^2)$         b.  $\Theta(2n)$
4. a.  $O(2^n)$         b.  $\Theta(n^2 2^n)$
5. a.  $O(n+5)$       b.  $\Theta(n \lg n)$
6. a.  $O(n \lg n)$     b.  $\Theta(2n^3)$
7. a.  $O(n+n^2)$     b.  $\Theta(n!)$
8. a.  $O(2n)$         b.  $\Theta(2^n)$
9. a.  $O(n-10)$      b.  $\Theta(n^2 + n)$
10. a.  $O(n!)$         b.  $\Theta(1)$

### C. Analyze solutions

See pseudocode you wrote for part C of the Intro assignment (same as part C of Assignment 2). Restate the problem.

For each of the numbered problems, the solution is in four parts, (a) to (d).

For an example of how to solve this kind of problem, see the handout sheet, "Sample problems in algorithm analysis."

- a. Write a recursive version of this algorithm.
- b. Use a *recurrence* to define the function that corresponds to the problem specification.
- c. Write a recurrence, based on (b), that defines the function that returns the algorithm's *running time* for input of size  $n$ .
- d. Solve the time recurrence, giving a one-sentence explanation.

### D. Design and analyze solutions

Do the same as in part C (a, b, c, d) for the problem matching your classroom ID among the following problems, where  $A$  is an input array and the return value is:

0. the largest element of a specified subsequence of  $A$ . Signature:  $Max(A, first, last)$ , where  $first$  and  $last$  are subscripts. Algorithm should return the largest element of the sequence from  $A[first]$  to  $A[last]$ , inclusive.
1. the smallest value in  $A$
2. *true* iff some two consecutive elements of  $A$  are identical
3. *true* iff  $A$  is in ascending order
4. the sum of the elements of  $A$
5. the sum of the linear series from  $x_1$  to  $x_2$
6. whether all the values in  $A$  are the same
7. the bitwise AND of bit vectors  $A$  and  $B$
8. whether there are at least two bits in a row that are identical in bit vector  $A$
9. the bitwise logical negation of bit vector  $A$
10. the number of occurrences of value  $x$  in array  $A$

### E. Brute force

See Study Questions, Topic 3, Longer Answer, "Brute Force problems."

For each problem to be solved:

- (a) Write a brute-force algorithm to solve the problem using iterative pseudocode, giving references if appropriate.
- (b) Write a recurrence that defines the function computed. For nested loops, this may involve writing two recurrences.
- (c) Write the corresponding time recurrence
- (d) Solve the time recurrence to give an analysis.

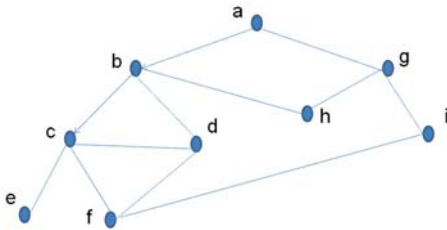
## Assignment for topic 4 (Divide and conquer)

Solve one of the problems in each set below.  
Restate problem and give letter and number.

### A. Graph search

- Give the order of a *depth-first* search of the graph below, starting at the vertex corresponding to your classroom ID.
- Give the order of a *breadth-first* search.

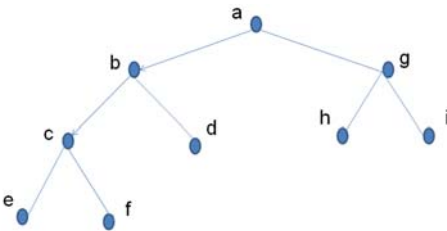
Please choose adjacent vertices in alphabetical order for ease of checking answers



Start at vertex:

- |      |                       |
|------|-----------------------|
| 0. a | 5. f                  |
| 1. b | 6. g                  |
| 2. c | 7. h                  |
| 3. d | 8. i                  |
| 4. e | 9. j (connects to g)  |
|      | 10. k (connects to f) |

### B. Tree traversal



List the order of visiting vertices for an traversal of the tree above, as follows. Explain running time.

- 1, 4, 7: preorder  
2, 5, 8: inorder

3, 6, 9, 0: postorder

### C. Recurrences and analysis

Referring to the pseudocode in the slides or textbook for the divide-and-conquer algorithms below, corresponding to your classroom ID.

- Write recurrence for the function computed
- Write time recurrence and give running time.
- State whether analysis is for worst or average case

Algorithms:

- Binary search
- BST search
- Heapify
- Order statistic (*k*th highest element of unsorted array) in time  $O(\lg n)$
- Merge sort
- Quicksort
- BST insertion
- Heap *extract-min*
- Heap sort
- Heap insertion
- Shortest-path for unweighted graphs based on breadth-first search (Johnsonbaugh, p. 188, Ex. 22).

### D. Coding and performance testing (Extra credit)

Code and performance-test the algorithms in part C, comparing your empirical results to the analysis provided. Offer a possible explanation of any discrepancy. Comment your code.

## Assignment for topic 5 (Greedy algorithms, Dynamic programming)

Solve the problems in each set below corresponding to your classroom ID. Give problem letter and number, restating problem.

### A. Greedy approach

Label the following as *algorithms* or *problems*, discussing your own words how the optimal substructure property applies where appropriate.

0. Prim's
1. Dijkstra's
2. Kruskal's
3. single-source shortest path
4. minimal spanning tree
5. knapsack
6. breadth-first search
7. fibonacci
8. change making
9. Floyd's
10. tree for prefix-free compression coding

### B. Greedy approach

Solve a problem from the Study Questions, topic 5, Longer Answer, corresponding to your classroom ID.

### C. Dynamic programming; transform and conquer

Solve the problem in each set below corresponding to your classroom ID. Groups may discuss solutions. Give problem letter and number, restating problem.

0. Explain Floyd's algorithm for reachability and its running time.
1. Write an algorithm to solve sorting-by-counting problem (see slide). Give a time analysis.
2. What are the space and time considerations raised by B trees?
3. Find and explain complexities of  $C(n, k)$  binomial coefficient algorithms that use (a) recursion (brute force) and (b) dynamic programming.
4. Describe an algorithm to find the *mode* (most frequent element) of an array. Analyze.
5. What are AVL trees used for, and how? Code the algorithm and give its time analysis.
6. Describe how the transform-and-conquer concept is used in AVL trees.
7. What space/time tradeoffs apply with hashing algorithms?
8. Compare the time complexities of the search algorithms for hash tables stored with open addressing (linear probe algorithm) and buckets (lists). State the range of possible load-factor values for each implementation and explain.
9. Needleman-Wunsch sequence alignment
10. Compare the time complexities of double-recursive Fibonacci and Fibonacci with dynamic programming.
11. Binomial coefficient.
12. Optimal BSTs
13. Longest common subsequence
14. Knapsack problem solution with table

## Assignment for topic 6 (Intractability)

Solve the problem in each set below corresponding to your classroom ID. Give problem letter and number, restating problem.

### A. Complexity classes

Explain the relationships between the following pairs of sets.

0. P and NP
1. P and EXP
2. NP and NPC
3. NP and the set of NP-hard problems (NPH)
4. NPC and NPH
5. EXP and NPH
6. P and  $O(n)$
7. NPC and  $O(2^n)$
8. NP-Hard and  $O(2^n)$
9. Intractable and NPC
10. Tractable problems and NP problems

### B. Problem complexities

Solve the problem corresponding to your classroom ID in the Study Questions, topic 6, Longer Answer, “6a. Complexity classes”.

### C. Reducibility

Explain what the following assertion means, and why you agree or disagree with it.

0.  $SAT \leq TSP$
1.  $SAT \leq \text{sorting}$
2.  $\text{sorting} \leq \text{searching}$
3.  $\text{knapsack} \leq \text{array search}$
4.  $\text{shortest path} \leq \text{minimum spanning tree}$
5.  $\text{Hamiltonian path} \leq TSP$
6.  $3SAT \leq SAT$
7.  $\text{Hamiltonian path} \leq \text{graph coloring}$
8.  $\text{knapsack} \leq \text{independent set of vertices}$
9.  $\text{set partition} \leq \text{Hamiltonian path}$
10.  $\text{set partition} \leq \text{minimum spanning tree}$

### D. Approximation

0. For what problems are approximation algorithms indispensable? Give an example.
1. When are heuristics used?
2. What does searching state spaces have to do with solving problems?
3. Describe the *generate-and-test* algorithm and the set of problems it solves. When is it unlikely to find an optimal solution?
4. What is *hill climbing* and why does it often find approximate rather than exact optima?
5. Compare *backtracking* to *branch-and-bound*.
6. What is the complexity of *minimax*, relative to the number of moves ahead it looks in a game, and why? Explain the algorithm as part of your answer.
7. What is *bounded rationality*?

### E. Randomized algorithms

Describe a randomized algorithm to *approximately* solve the following, and give running time, citing sources if appropriate:

0. the stable-marriage problem
1. finding a maximum matching in a bipartite graph
2. traveling salesperson
3. SAT
4. 3SAT
5. Hamiltonian path
6. set partition
7. graph coloring
8. function optimization on  $n$  variable

## Assignment for topic 7 (Parallel-distributed; interaction)

Solve the problem in each set below corresponding to your classroom ID. Give problem letter and number, restating problem. Give sources consulted if appropriate.

### A. Short-answer questions

Solve the problem in Study Questions, topic 7, “Short and Longer Answer Questions”, corresponding to your classroom ID.

### B. Parallel algorithm discussion

Describe an efficient parallel algorithm to find the following, for an array of integers, giving parallel running time. Justify your analysis.

0. the maximum value
1. the number of 0's
2. the product of all the values
3. the number of values greater than 2
4. the sum
5. whether or not the array is sorted ascending
6. the minimum value
7. the average
8. the median
9. the AND

### C. Parallel algorithm design

(a) Write a parallel algorithm, using *parallel for* in pseudocode, to find the following for an array of  $n$  integers

(b) State parallel time and work complexity and justify your answer.

0. whether all values are the same
1. set all values to 0
2. add 1 to all values
3. the OR, if the integers are all 0 or 1 (return 1 iff at least 1 element of  $A$  is 1)
4. the AND, if the integers are all 0 or 1 (return 1 iff at all elements of  $A$  are 1)
5. the bitwise negation
6. the OR of *two* arrays
7. the AND of *two* arrays
8. the index of an element of array  $A$  that matches *key*
9. replace all values greater than 1 with 0

### D. Algorithms vs. interaction

0. Summarize Peter Wegner's view of reasoning versus interaction.
1. What sort of computation is associated with computing functions, and what sort of computation is associated with providing services?
2. Distinguish two types of interaction by the number of active agents involved.
3. Distinguish two types of interaction by the use or non-use of the environment in interaction.
4. Argue that interaction is more powerful than algorithms.
5. Argue that interaction is *not* more powerful than algorithms.
6. Argue that multi-stream interaction is more powerful than sequential interaction; that it is not more powerful.
7. Whereas algorithmic computation operates on strings and natural numbers, what does sequential interaction operate on? Explain.

### E. Performance of interactive processes

0. What measures of efficiency are applied in interactive computing?
1. In multi-agent systems, what scalability issues are raised by the choice between message passing and shared access to storage?
2. Compare scalability of algorithms with scalability of multi-agent systems.
3. Discuss the time complexity of sequential interactive processes.
4. In multi-agent systems, what scalability issues are raised by the choice between message passing and shared access to storage?

## Assignment for Summary

Referring to the pseudocode in the slides or textbook for the divide-and-conquer, greedy, or dynamic-programming algorithms below,

- a. Use loop invariant to argue for correctness
- b. Write recurrence for the function computed
- c. Write time recurrence and give running time.

*Algorithms:*

0. Binomial coefficient  $C(n, k)$  algorithm that uses dynamic programming.
1. Heapify
2. Order statistic ( $k$ th highest element of unsorted array) in time  $O(\lg n)$
3. Merge sort
4. Minimal spanning tree
5. Convex hull
6. Closest pair
7. Heap insertion
8. Quicksort
9. Warshall's algorithm
10. Greedy graph coloring
11. Dijkstra
12. BST sort